A Differential Approach to Graphical Interaction

Michael L. Gleicher

November 18, 1994
CMU-CS-94-217

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Andrew Witkin, Chair
Paul Heckbert
Brad Myers
Robert Sproull, Sun Microsystems

This research was supported in part by Apple Computer, an equipment grant from Silicon Graphics Inc., and a fellowship from the Schlumberger Foundation. The views and conclusions contained in this document are those of the author and should not be interpretted as representing the official policies, either expressed or implied, of these companies.

Keywords: Constraints, Direct Manipulation, Interaction Techniques, User Interface Toolkits

Abstract

Direct manipulation has become the preferred interface for controlling graphical objects. Despite its success, the ad hoc manner with which such interfaces have been designed and implemented restricts the types of interactive controls. This dissertation presents a new approach that provides a systematic method for implementing flexible, combinable interactive controls. This *differential approach* to graphical interaction uses constrained optimization to couple user controls to graphical objects in a manner that permits a variety of controls to be freely combined. The differential approach provides a new set of abstractions that enable new types of interaction techniques and new ways of modularizing applications.

The differential approach views graphical object manipulation as an equation solving problem: Given the desired values for the user specified controls, find a configuration of the graphical objects that meet these constraints. To solve these equations in a sufficiently general manner, the differential approach controls the motion of the objects over time. At any instant in time, controls specify desired rates of change that form linear constraints on the time derivatives of the parameters. An optimization objective selects a particular value when these constraints do not determine a unique solution. The differential approach solves these constrained optimization problems to compute the derivatives of the parameters. An ordinary differential equation solver uses these rates to compute object motions.

This thesis addresses the issues in using numerical techniques to provide interactive control of graphical objects. Techniques are presented to solve the constrained optimization problems efficiently and to dynamically define equations in response to system events. The thesis introduces an architecture, called Snap-Together Mathematics, that encapsulates these numerical needs. A graphics toolkit, constructed with Snap-Together Mathematics, provides the features of the differential approach yet hides the underlying machinery from the applications programmer.

The thesis demonstrates the differential approach by applying it to a variety of interaction problems, including manipulation of 2D and 3D objects, lighting, and camera control. Demonstrated interaction techniques include novel methods for some specific interaction tasks. A number of prototype applications, including 3D object construction and mechanisms sketching, demonstrate the tools and the approach.

If I lost my mind, would you help me find it?
If I lost my mind, would I have to be reminded?

— Soul Assylum
Spinning

Acknowledgements

I acknowledge everyone who needs acknowledged.

With so many other pieces of thesis to work on, I'm tempted to leave it at that. But, thanks to a large number of people, my six years in Pittsburgh have left me with a lot more than just the regional dialect.

It would be a lie for me to say I don't know where to begin. First I would like to thank my parents for their love and support throughout the years. The ski trips to Colorado the past few years were particularly useful in helping me keep my sanity as the throes of graduate student life stressed me out. My sister, grandmothers and Uncle Robert were all particularly understanding that my schedule made visits infrequent.

My six years at CMU have been a wonderful opportunity to learn and grow, not just as a computer graphics researcher, but as a person in general. Surviving the experience would not have been possible without a great set of friends who were always there to help me through the hard times, and to celebrate the good. Scott Nettles was there from our first attempts to figure out how to buy beer under Pennsylvania's laws to the celebrations as I finished. He always provided a willing ear for my complaining. Bryan Loyall and Peter Weyhrauch, my housemates for the past 5 years, helped make the house on S. Atlantic Ave. a great place to call home. Bruce Horn and Spiro Michaylov suffered through innumerable early drafts of my papers and still hung around for the fun things afterwards. It's impossible to list everyone, but David Steere, Lin Chase, James Landay, Jim Blythe, Phyllis Ruether and Greg Morrisett are the first people I think of.

Ian Davis encouraged me to get back to playing music, a much needed diversion. He, Shaun McDermott, and the rest of Painted Mice provided an outlet for me to do something besides computer science. The Thursday dinner club helped keep me well nourished, nutritionally and intellectually. And a special thanks to Lori Fabrizio for being special and for her care and patience over the past 2 years.

My advisor, Andy Witkin, gave me countless good ideas, talked me out of a lot of bad ones (and tried to talk me out of some good ones as well), and was patient with me as I learned to do math and write. He and the rest of my committee, Paul Heckbert,

Brad Myers, and Bob Sproull, really helped me turn a jumble of ideas into something resembling a thesis. Will Welch, my officemate and co-conspiritor for the past 5 years, shared countless amounts of caffeine and conversation, and in the process gave me an amazing amount of mathematical intuitions. David Baraff, Sebastian Grassia, Paul Heckbert, and Zoran Popovich all helped make the 4th floor of Doherty Hall an exciting place to do computer graphics. Phyllis Pommerantz was our "den mother." And no CMU CS thesis would be complete without thanking Sharon Burks and Catherine Copetas who really make the place run.

One of the most fun aspects of doing this thesis was to become part of the worldwide computer graphics research community. I'd like to thank everyone who shared ideas, encouragement, and skepticism. I would especially like to thank everyone at the graphics group at Apple ATG, which was my home away from home for two summers. A special thank you for the loaner computer to help with the thesis writing.

Writing this is a lot harder than I had expected. It's difficult to summarize six years of great experiences on one page. I guess I took two, and still only scratched the surface.

Contents

List of Figures

and as she stepped from out her shell
and looked around for luck;
"Quack," said Jerusha,
"I seem to be a duck."

— Mildred P. Merryman
"Quack!" said Jerusha[Mer50]

Chapter 1

Introduction

Ever since computers have had graphical displays and pointing devices, graphical manipulation has been an important means of communicating between people and computers. Such interfaces couple the behavior of some graphical object to the input device, continuously tracking its changes with motion. Sketchpad [Sut63], the earliest interactive graphical application, introduced this style of interface, which has come to be known as direct manipulation.[1] Input and output devices continue to evolve from Sketchpad's vector display and light pen. Yet after 30 years of advancements in the hardware for interfaces, the basic notion of direct graphical manipulation remains the same.

As computers capable of supporting direct graphical manipulation have become more common, it has become the dominant interaction method for configuring graphical objects. However, present approaches to realizing graphical manipulation severely limit the types of interfaces which can be constructed. They restrict the types of interactive controls that can be provided to users and provide no facilities for combining these controls.

This thesis considers how the numerical and graphical performance of modern computers can be exploited to create an approach to realizing graphical manipulation that avoids the limitations of previous approaches. I will introduce a *differential approach* to graphical interaction, in which constrained optimization is used to couple the motion of graphical objects to a user's controls. To create such an approach to graphical interaction, we must consider what types of mathematical techniques to employ, what interaction techniques to build with them, and how to incorporate them into interactive applications.

[1] Although the term "direct manipulation" is generally attributed to Ben Schneiderman [Sch83], the ideas predates his work.

1.1 Implementing Graphical Manipulation

Direct manipulation has become the dominant style of graphical interaction with good reason: it provides a uniform mode of interaction that resembles interaction with real objects in the real world. The controls on a graphical object are handles that the user can grab and drag. As the user drags a handle, the object follows the motion of the pointing device with continuous motion, providing kinesthetic correspondence.

The success of graphical manipulation leads to a desire to extend its range to a wider variety of graphical objects, control types, and applications. However, present approaches to implementing graphical manipulation limit this range. The task of implementing direct manipulation requires mapping from the user's actions on the handle to changes in the program's internal representation of the object and providing feedback to the user of these changes. To date, the former has been implemented in an ad-hoc manner. Each new type of handle must be specifically hand-crafted.

Hand-crafting each handle places two significant restrictions on the types of interfaces that can be created. First, it restricts the types of handles to those for which the mapping to object parameters can be determined by the programmer. Second, it restricts how handles can be combined, as any combination must also be hand-crafted. Because there is no standardized mechanism for defining the mappings between handles and parameters, defining new types of handles can be a difficult task.

To better illustrate these problems, consider a simple example: positioning a line segment in a drawing program. Even with this simple graphical object, there are many attributes that the user might want to specify, such as the positions of the endpoints, the position of the center, the length or the orientation. Ideally, the program should permit the user to control directly whichever attribute they desired and and mix-and-match these controls as needed. That is, each attribute should have an associated handle so that the user can select controls that are most convenient to their task, and a user should be able to employ multiple, simultaneous controls to more fully specify their intents.

A simple way for a program to represent the line segment is to store its two endpoints. This representation makes it is easy to position an endpoint: simply set a pair of parameters equal to the position of the mouse. Providing other handles is more difficult. For example, to permit the user to manipulate the length of the line segment directly requires the interface implementor to work out a bit of mathematics to compute the positions of the endpoints from the length. Had a different representation been chosen, implementing this control would have been easier. For example, if the programmer had chosen to store the center, orientation and length of the line segment, the set of attributes that could easily serve as handles would be different.

With the ad-hoc implementation methods, simultaneous controls, either to support multiple input devices or to express constraints on the object changes, require explicit hand-crafting of each combination of controls. For example, maintaining the position of one endpoint of the line segment while the other is dragged can be implemented

easily if the line is represented by the positions of its endpoints. However, an interaction that maintained an endpoint's position while the center of the line segment is dragged would require some mathematical work by the interface designer if either of the representations from the previous paragraph were used.

For an object as simple as the line segment, it might be possible to predict all possible combinations of controls, or at least a sufficient set of possible combinations. However, combinatorics makes this impractical with more complicated objects. Similarly, if we consider simultaneous control of multiple objects, the increased combinatorial possibilities make explicit coding of all combinations impossible. Controls on multiple objects, such as relative positions or differences in size, further compound the problem with more potential handles, more possible combinations, and less possibility of predicting what the user will need.

Without a general mechanism for defining the mapping from a handle to the object's parameters, it is difficult to define new handles and combinations. As a result, all combinations of controls must be pre-designed by the program implementor, making experimentation with combinations of controls difficult, and dynamic combination of controls by the user impossible.

Even if combinatorics do not make it impossible to switch representations to provide alternate combinations of controls, other issues limit possible interfaces with the approach. Often, concerns such as numerical stability, freedom from singularities, and implementation convenience restrict the representations that can be used for objects. The tension between these implementation concerns and user needs leads to interfaces where the users must manipulate non-intuitive, but mathematically convenient, controls, such as B-Spline knot points, or suffer with inferior representations, such as the singularity-ridden Euler angles used by many systems for storing 3D orientations [Sho85].

In summary, the ad-hoc methods previously used to implement direct manipulation have many problems. As shown in the examples of the preceding paragraphs, they

- limit the types of interactive controls that can be provided to users;
- prevent interactive controls from being freely combined as desired;
- restrict the types of representations that programmers can use inside systems to those that user controls can be conveniently mapped to;
- fail to provide a consistent set of abstractions for defining interaction techniques;
- fail to provide a methodology for defining new controls, making it difficult to experiment with new ideas;
- prevent the realization of some potentially desirable interface styles.

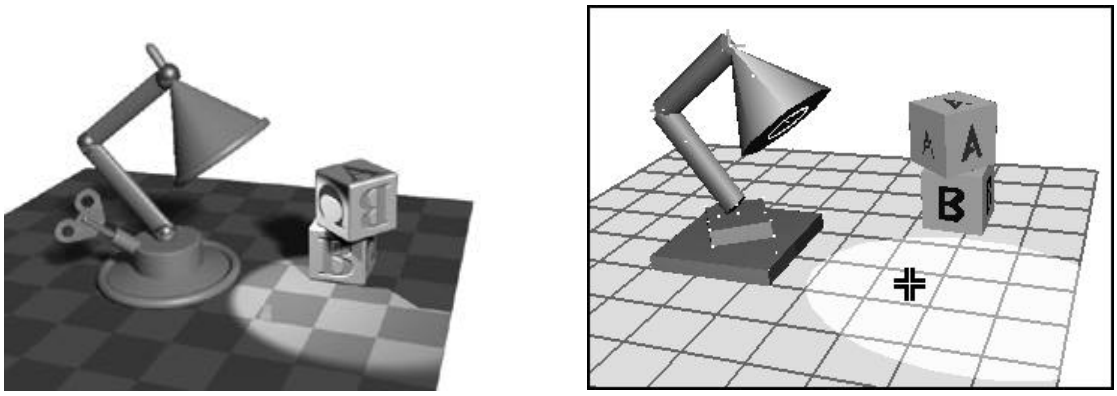

Figure 1.1: A 3D scene in which a Luxo lamp is used not only as an object in the scene, but also for illumination. To create this image, the user must configure the lamp so that the light falls in the desired location. The techniques of this thesis allow the user to control the lamp by manipulating the light's target directly, and have the lamp be adjusted accordingly. The right image shows an interactive scene composition system, described in Section 9.7, being used in this manner. (Thanks to Drew Olbrich for the ray tracing.)

This thesis provides a systematic approach to implementing direct graphical manipulation in a way that avoids these problems while keeping the essential benefits of direct manipulation.

1.1.1 A Systematic Approach to Realizing Graphical Manipulation

Our goal is to have flexible interactive controls that can be freely combined. For some interfaces, this increased power might be provided directly to users who could mix and match controls as needed for their problem. However, the extra power also helps users indirectly by giving interface designers more choices in what they can provide to users.

Some of the benefits of this flexibility are illustrated in the example of Figure 1.1. Consider an interactive application that allows a user to manipulate desktop objects, for example to create pictures of office scenes. There are many things a user may want to do with the lamp, for instance, they might want the light to shine onto a particular place, place the lamp in a particular position, or orient the lamp a certain way.

Inside the application, the configuration of the lamp might be represented as the position of the base and the angles of each of the joints, or it might be represented as the position and orientation of each part of the lamp. The former is preferable because it maintains the connections between parts of the lamp. Unfortunately, to implement a handle that permits the user to grab and drag the lightbulb, a programmer must somehow devise a mechanism to update the joint angles accordingly. The ad-hoc approaches to realizing graphical manipulation give little help in deriving such mathematics. Because the effort of deriving the handle's implementation would most likely be very

specific to the Luxo lamp, traditional[2] direct manipulation systems would most likely be forced to provide the user with only direct control over the joint angles. While this is sufficient to configure the lamp, it is not necessarily convenient for tasks like positioning the light bulb or aiming the light.

This thesis presents a systematic approach for implementing direct graphical manipulation. A general-purpose mechanism maps between the handles provided to the user and the parameters of the graphical objects. With such an approach, a user of the Luxo lamp example could not only manipulate the joint angles, but could also grab any part of the lamp directly, The programmer did not have to explicitly code the mathematics to map the manipulations into parameter changes. In fact, the flexibility in controls permits definition of other less obvious handles that permit the user to have direct control over attributes of interest. For example, a user interested in shining the light onto a particular location could simple grab the target of the lamp (the center of the spot) and drag it to the desired location. The controls can be freely combined. For example, a user could position the light's target and simultaneously specify the lamp's position on the table.

A systematic approach to realizing direct manipulation can be based on a general purpose mechanism for mapping user controls to object parameters. Creation of such a mechanism requires us to view graphical manipulation as a constrained optimization problem. To solve this problem in a practical manner, we must treat it *differentially*, that is to control how objects change rather than their final targets. This thesis introduces a *differential approach* to graphical interaction that begins by taking the view of manipulation as a mathematical problem. To realize the approach, the thesis will provide mathematical techniques to solve the problem, implementation techniques to address pragmatic issues, and a system architecture to use the approach to build applications. Example interaction techniques will be provided to show the promise of the approach, and applications will be demonstrated to show its viability.

1.1.2 Classes of Users and Tools

There are different classes of people involved with an interactive graphical application. As in Myers' survey [Mye93], we will need to distinguish these into distinct categories. Myers' categories are users, interface designers, application programmers, and tool creators. For the purposes of this thesis, we will lump interface designers and application programmers together as their tasks are similar: to build the application that the user will employ in their graphical task with the tools provided by the tool creators. The application builders will be the users of application development tools, but unless we

[2]The Luxo lamp is an example of an important special case: an articulated figure. Recently, several commercial animation systems, such as SoftImage [Sof93] and Wavefront [Wav94], have included inverse kinematic techniques to manipulate such objects by positioning end-effector points. These methods, and their limitations, will be reviewed in Section 2.2.4.

explicitly refer to the "user of the toolkit," the term "user" will refer to the "end user" of the graphical application.

The work of this thesis affects all three groups. While our approach can be employed to provide conventional interfaces, it may also be used to provide new types of interfaces for users. It gives the application programmer a new set of abstractions with which to build interactive systems. Finally, for the toolkit builder, there is a new class of services that must be provided, but these services can help enhance the modularity of the tools by: providing a standard interconnection mechanism between objects; allowing the internal representation of the objects to be hidden from applications programmers; allowing tools to be provided to the application programmer that allow pieces to be assembled by combination and composition to form interaction techniques; and allowing the encapsulation of numerical constraint computations.

One might consider applications where the user is exposed to the mathematics behind their graphical application. For example, the CONDOR system [Kas92] allows the user to construct mathematical expressions that define the graphical objects. Although such an application can be constructed using the approach of this thesis, this thesis focuses on applications where the user is insulated from the mathematics, instead directly manipulating graphical objects. In fact, a goal of this thesis is to hide as much of the mathematics as possible inside the applications development tools so that only the tool creators need see it.

1.1.3 Graphical Manipulation as Equation Solving

To introduce the differential approach of this thesis, graphical manipulation must be viewed as a constrained optimization problem. Graphical manipulation deals with how a user configures a set of graphical objects to achieve some desired goals. For the lamp example, the set of graphical objects consists of the Luxo lamp, the table top, and the other objects on the table such as the blocks. I will often use the term *model* to refer to the set of objects.

In the class of graphical manipulation tasks considered in this thesis, users manipulate objects whose configurations can be stored as a concise set of real-valued parameters, called the object's *state vector.* For a given object, there are potentially many sets of parameters which might equivalently serve as a representation, as demonstrated by the line segment example. A *parameterization* is a particular representation of the state of an object.

Objects usually have many attributes that may be of interest to an observer. Since, by definition, the state vector fully describes the configuration of the object, the attributes must be determined as functions of these parameters. For this thesis, we restrict ourselves to the broad class of object attributes which can be computed by closed-form, differentiable expressions over the state variables. This class includes many of the types of models used in interactive computer graphics such as most parametric and

implicit curve and surface representations, transformation hierarchies, virtual cameras, and many simple shading models. We will not consider things such as combinatorial or discrete attributes, such as the number of sides of a polygon, or attributes computed by recursive or iterated functions such as fractals.

A *control* is an attribute of an object that can be specified or directly manipulated. For example, if a system allowed the user to drag the position of the lamp's lightbulb or the target location of the light, these attributes of the lamp would be serving as controls. A *constraint* is a control for which a fixed value is given, preventing the value of the attribute from changing. Such controls constrain the behavior of objects by restricting their motion so that the constraint is not violated. For the purposes of this thesis, the terms constraint and control are nearly interchangeable: a constraint is a control with its value fixed, a control is a constraint whose value is being specified dynamically by the user, e.g. a value constrained to follow the mouse.

A single control generally does not uniquely determine a configuration of the object. For example, if one endpoint of a line segment is specified, there is still a continuum of possible configurations for the segment. To combat such *under-constrained* situations, it is often desirable to use multiple controls simultaneously. In the cases where there is only a single input device, dragging manipulation might be combined with constraints (e.g. controls that are restricted from changing). In a sense, even a single dragging operation can be thought of as multiple controls if we consider each axis of the pointing device independently.

It is unreasonable to require the user to employ enough controls to uniquely determine the configuration of the graphical objects. The user simply may not know or care about some attributes of some objects, or it might be too much work to specify everything. In such under-constrained cases, the system must somehow choose one of the possible configurations. Without mind reading, it is impossible to reliably select the solution that the user most desires. Systems must settle for simply trying to select a solution that is reasonable. One version of this is the "Principle of Least Astonishment" [BDFB$^+$87] which suggests the system should try to select the option that will surprise the user the least.

For an analogy, think of a model as a large machine which has a few knobs for the user to turn and many gauges whose values the user may be interested in. Suppose there are a few gauges for which the user desires a particular value. The graphical manipulation task is to find settings of the knobs such that the gauges reach these desired values. If each gauge to be specified corresponds directly to a knob, the task is easy, because each knob can be turned and set independently. However, most gauges will depend on complicated combinations of the knobs, making it harder to find settings of the knobs that achieve desired values. In this metaphor, the knobs are the parameters of the graphical objects, the gauges are attributes of the objects that the user may be interested in, and the internals of the machine correspond to the functions that compute the attributes from the parameters. Traditional implementations of direct manipulation

require the user to control the knobs directly. The methods of this thesis permit the user to use any of the gauges as controls by automatically adjusting the knobs as needed.

1.1.4 Goals for Graphical Manipulation

Treating interactive control as the specification of values for controls as in the last section leads to a concise mathematical problem. The user would like to specify some set of controls, $\mathbf{p}$. The system needs to find some configuration of the state variables, $\mathbf{q}$, which meets this. Since the controls can be computed as a function of the state variables, we have

$$\mathbf{p} = \mathbf{f}(\mathbf{q}). \tag{1.1}$$

Solving the manipulation problem is, at one level, as straightforward as solving this equation for $\mathbf{q}$. However, there are many difficult goals which we might want our solution technique to meet:

1. flexibility in the types of controls, and therefore the functions which compute them;
2. freedom to combine controls arbitrarily, "mixing-and-matching" them dynamically;
3. keeping the good properties of direct manipulation, e.g. continuous motion, rapid feedback, tight coupling of the input device to objects on the screen, . . . ; [Sch83]
4. choosing the "best" solution in under-constrained cases, and finding a "reasonable" answer even if there is no exact solution.

To aid the application implementor, there are several other goals:

5. freedom in picking representations independently of user concerns;
6. a standard procedure for defining new controls that minimizes the amount of difficult mathematical work in defining a new type of control;
7. a solving mechanism that is general purpose and encapsulatable so that a single common implementation can serve a number of applications and so that the application developers need not worry about the details of the solving mechanisms.

We would like the techniques developed to realize the approach to also:

8. work over a variety of domains;
9. be fast and scale well;
10. require only readily available, easy to code numerical algorithms. Reliance on sophisticated numerical codes that must be purchased from commercial vendors or developed by expert numerical analysts would be unacceptable.

1.1.5 The Problems of Other Approaches

Our goals make solving Equation 1.1 for **q** impractical for three general reasons:

- in order to have flexibility in the types of controls, non-linear equations may need to be solved. Such equations are hard to solve;
- in order to have flexibility in the number of controls that are specified, we must permit under-constrained and over-constrained cases;
- in order to provide the desired direct manipulation interface, object must move with continuous motion. Therefore, the solver must be fast enough and provide continuity in the solutions.

In order to provide direct manipulation with general controls by solving Equation 1.1, we must solve arbitrary systems of non-linear equations fast enough to allow for frequent enough updates to give the user the illusion of continuous motion.

In order to meet goal 1, the equation solver must be able to handle a wide range of functions, including non-linear equations. Without knowledge about the functions to be solved, sets of equations are difficult to solve. Not only is good information hard to find in general, but each combination of equations might also require specific knowledge. Because of this, [PFTV86, Chapter 9] argues that not only does no reliable, general, non-linear solver exist, but that one cannot exist.

Without global information about functions and combinations, solving techniques must rely on local information, effectively searching for solutions. Almost all non-linear solvers are iterative methods that take an initial guess as to the solution and repeatedly update the guess until they find a solution. Such a solver can never determine that there is not a solution: if it fails to find a solution it might simply mean that it has not searched hard enough. These solvers will be discussed further in Section 2.2.2.

As computers grow faster, it might become practical to consider using a sophisticated non-linear equation solver to provide direct manipulation. However, such an approach is unlikely to succeed for a number of reasons:

- despite their sophistication, the methods are heuristic and not completely reliable;
- because they are doing searches, it is difficult to predict how long it will take them to find a solution;
- the solvers may fail to find a solution, but only after spending a long time looking for it;
- the solvers do not degrade gracefully: it is difficult to limit the amount of time that they spend because their intermediate states may not be close to the answer;

When we examine the previous approaches to implementing graphical manipulation, we see that they all fail to meet some of these goals. Previous work will be explored in more detail in Chapter 2.

Traditional Direct Manipulation – The traditional method for implementing direct graphical manipulation has been to couple parameters directly to the pointing device. For example, with the luxo lamp, a conventional direct manipulation system would allow the user to connect a joint angle to a knob. Some mappings between the input and the values are possible, for example to convert the linear motion of a slider to the rotary motion of the joint, but there must be some direct way of computing the parameter values from the inputs.

Traditional implementations have been the mainstay of direct manipulation interfaces. Such interfaces have been very successful, largely due the fact that it meets goals 3 and 9. However, its limitations have restricted the types of interfaces that have been constructed. Traditional direct manipulation severely restricts the types of functions which can be used as controls (goal 1) and it provides no automatic way to combine controls (goal 2). Parameters must be chosen so that the controls will map onto them easily (failing goal 5). Because good representations must be developed for any new controls, and because these closed form mapping for controls must be found, developing new controls can be difficult work (violating goal 6).

Parametric Modeling Approaches – Parametric modeling is a variant of the traditional direct manipulation approach. Such schemes permit end users to create models with parameter dependencies. These parameters are directly specified. Parametric approaches permit a clever user to overcome some of the deficiencies of the traditional direct approach. For example, if the designer of the Luxo lamp knew the user would want to control the height of the lamp, but not the joint angles, they might have devised a way of representing the configuration of the lamp so that height is a parameter, and the joint angles are computed from that. Parametric approaches suffer from the same failures of direct manipulation, although it does permit a clever user to sometimes have some additional flexibility in the types of controls.

Traditional Constraint-Based Approaches – A constraint-based interface[3] treats Equation 1.1 by employing an equation solver. Typically, the user specifies values for various aspects of the model and then the system solves for some value of the state vector which meets these constraints. We call such a constraint-based approach a "specify-then-solve" style.

[3] I use the term *constraint-based* interface to mean that constraints are an abstraction provided to the end user of a system, rather than simply as an abstraction used by programmers.

Although the problem of solving the equations required to meet goals 1 and 2 is difficult, a bigger problem with a specify-then-solve approach is that it fails to meet goal 3. After the user specifies the constraints, the system solves the equations and then displays the result to the user. Objects jump to the new configuration, leaving the user to puzzle out what happened. This makes goal 4 even more difficult. It becomes critical to pick a good solution to avoid confusing the user. Picking the correct solution is also important because without the rapid feedback of direct manipulation it can be difficult to explore possible solutions.

A system designer might consider using interpolation to provide the desired continuous motion in a constraint solving system. After solving for a new configuration a system might make a smooth transition by interpolating between the old state and the new. However, jumping between configurations cannot be avoiding by simply interpolating. Unless something enforces the constraints in the intermediate states, the constraints may be broken, leading to potentially undesirable behavior.

Specialized Constraint-Based Approaches – The primary drawback of the traditional constraint-based approach is that it violates goal 3, the desire for direct manipulation. One approach to handling this is to restrict the class of constraints so that they can be solved faster. The best examples of this are the propagation constraint solvers, such as DeltaBlue [FBMB90]. In essence, these algorithms trade-off goals 1 and 2, in order to better meet goal 3. As a side effect, some of these algorithms provide techniques, such as constraint- hierarchies [BFBW92], to handle under-constrained cases (goal 4). Unfortunately, propagation solvers restrict the set of possible controls and the ways controls can be combined in ways that are unacceptable for graphical manipulation (failing goals 1 and 2). Also, for each new control, a variety of bi-directional methods must be generated, which may not be easy for many types of functions (failing goal 6).

The problem of determining configurations that achieve the desired attribute values is an important problem in robotics and computer animation. Such solving is referred to as *inverse kinematics*. The inverse kinematics literature, examined in Section 2.2.4, includes numerical methods that solve the systems of non-linear equations. A problems of particular interest to robotics, namely configuring articulated figures by positioning end-effectors, is particularly well-studied. Highly developed techniques have been developed and are commonplace enough to be surveyed in robotics textbooks, such as those by Craig [Cra86] or Paul [Pau81]. The techniques are now appearing in commercial computer animation systems, such as Softimage [Sof93] and Wavefront [Wav94]. The methods in such systems are not general: they only permit manipulation of a very specific control on a very specific class of model (failing goals 1, 5 and 8), and typically provide only a single control at a time (failing goal 2). The differential approach can be

viewed as a use of generalized inverse kinematics to create a general approach to implementing graphical manipulation.

1.2 The Differential Approach

Existing approaches fail to meet the goals for graphical manipulation, demanding the development of a new approach. An advantage that we have over the developers of previous approaches is that computer hardware has advanced to the point that the machines on which graphical applications are run have considerable computational and graphics performance. Such machines make it possible to do non-trivial numerical calculations in between each frame of continuous motion animation. This means that it is possible to perform some numerical constraint calculations and still provide a continuous-motion direct manipulation interface. This thesis presents such an approach to graphical interaction.

Our goals make solving the manipulation problem of Equation 1.1 difficult. Previous approaches have either restricted the equations, or restricted the desired direct graphical interaction. In this thesis, I will present an approach which makes a different kind of restriction: that we are interested only in direct graphical interaction and will always demand that objects move with continuous motion, not jump between very different configurations. The interfaces desired for graphical interaction have this property.

Because we are considering cases where objects move continuously, it is sufficient to control them by controlling how they change over time. By controlling how objects are changing, rather than controlling their configurations directly, a variant of Equation 1.1 may be solved. Controls specify the attributes' rates of change and the system solves for the state variables' rates of change to make this happen. I call this approach to graphical interaction based on this control by time derivatives the *differential approach.*

With the differential approach, at particular instants in time a solver must determine the time derivatives of the state vector given the time derivatives of the controls. We refer to this as *differential optimization.* Solving the differential optimization is a much more mathematically tractable problem than solving Equation 1.1 directly. This means that it is possible to provide direct graphical interaction (meeting goal 3), while handling a general class of non-linear functions (meeting goal 1), and allowing these to be combined in arbitrary ways (goal 2). Methods for solving the differential optimization problem address the issues of under-constrained and over-determined cases (goal 4).

The differential approach meets the implementation goals as well. By allowing almost arbitrary non-linear functions to map between controls and parameters, it provides flexibility in selecting representations of objects independently of how they will be manipulated (goal 5). The solving methods require little information about the control functions, in fact, all that is required can be found automatically given the control

function (goal 6). The mechanisms behind the differential approach are general purpose and can be encapsulated in a manner that not only hides the underlying mathematical techniques, but also permits a single implementation to serve as a building block for almost any type of system requiring graphical manipulation (goals 7 and 8). The techniques to realize the approach perform well enough to work on current machines (goal 9), without resorting to numerical routines beyond those in standard textbooks (goal 10).

1.2.1 Direct Manipulation in the Differential Approach

Digital computers provide the illusion of continuous motion of graphical objects by repeatedly redrawing the image. The time between these redraws must be sufficiently small in order for the illusion to be maintained. To support direct manipulation, a system must sample the position of the mouse and update the positions of the objects at a rapid rate.

The differential approach breaks the numerical constraint solving problem into two parts: computing the rates of change of the parameters at particular instants, and computing the trajectory of the parameters over time, given the rates of change at particular instants. The former problem is the differential optimization problem, and the latter is solving an ordinary differential equation (ODE). Between each redraw, the ODE must be solved to update the configurations of the graphical objects. Each of these solver steps advances the configuration by solving some number of differential optimizations, each determining the rate of change at some particular instant.

With the differential approach, the graphical objects cannot simply be moved with the mouse. Instead, each step they move towards a target. Limitations in ODE solving, discussed in Section 3.3, provide speed limits on how quickly objects can move, so they may not be able to reach their target in the time provided. If the target is the position of the mouse, this will cause the object to lag behind its target, gradually catching up as the mouse slows down. This can make manipulation feel as if the objects are connected to the input devices by springs, and will be discussed in Section 6.1.2. As computers grow faster, more computation can be done between each redraw while maintaining a rate sufficient to provide the illusion of continuous motion. This allows raising the effective speed limits of the objects, and can reduce the lag.

1.2.2 An Alternate View of Graphical Manipulation

An alternatative view of graphical manipulation is to imagine graphical objects as physical entities that are manipulated as physical objects in the real world: by pushing and pulling on them. With such a view, implementing direct manipulation becomes a problem of implementing an interactive physical simulation. The issues in creating such

simulations are explored by Witkin et al.[WGW90]. The techniques presented in that paper form the basis for this thesis.

The differential approach can be viewed as a variant of the physical simulation approach. The physics of the "world" is modified from that of the real world in order to facilitate manipulation. Most significantly, inertia is removed by replacing Newton's law of motion, $f = ma$, by its first derivative equivalent, $f = mv$. An object in motion is only in motion while it is being acted upon by a force. For manipulation, this has the advantage that objects remain where they are placed, rather than skidding around.

The mathematical methods used for implementing the differential approach presented in Chapter 3 are the same as those used for implementing physical simulations. Many of the numerical methods and implementation techniques in the thesis were originally conceived for implementing interactive simulations. Presenting the differential approach as constrained optimization, as done in this thesis, rather than presenting it as a physical simulation, is largely a matter of taste.

1.3 An Approach to Graphical Interaction

The ultimate goal of this research is to improve the quality of graphical manipulation interfaces. The central focus of this thesis makes an indirect step towards this goal, providing a new set of abstractions which provide more flexibility in the type of interaction techniques that can be created. This increased flexibility does not necessarily imply better interfaces — in fact, they give interface designers new ways to baffle and confuse users. However, there are several reasons to believe that the differential approach can lead to improved interaction techniques.

The differential approach permits building interfaces which have many desirable properties. It provides for continuous motion of the graphical objects. It permits interfaces to provide controls to the user which permit directly controlling attributes of interest. These controls need not directly connect to the parameters. It permits controls to be combined, either by the user or by interface elements.

The example interaction techniques of Chapter 8 show the promise of the approach. The examples which recreate prior techniques show that the abstractions provided by the differential approach are sufficiently rich to create usable interactions. Some of the newer techniques, such as the through-the-lens camera controls of Section 8.1.4 could not have been considered with previous approaches to building interfaces. Some of the examples, like the artificial horizon of Section 8.3.5, are not good interfaces. But with the differential approach, techniques can be explored without deriving the inverse mathematics, so it is possible to learn that they are unusable before investing a large amount of time and effort in their development.

The differential approach provides a new set of abstractions for building graphical interaction techniques. In the remainder of this section, we briefly introduce the

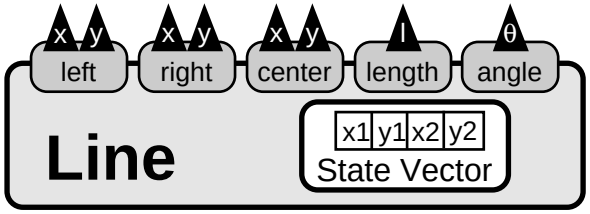

Figure 1.2: A schematic representation of a simple graphical object. The object stores a set of parameters internally in its *state vector.* However, the outside world accesses the object via its connectors, providing flexibility and parameter independence.

abstractions, along with the terminology used throughout the thesis.

1.3.1 Graphical Objects and Connectors

For the purposes of this thesis, we are concerned with what are commonly called *object-oriented* graphical editors. In such applications, the user deals with finite sets of graphical objects which must be manipulated to create the desired model or drawing.

For the most part, graphical objects are the visible entities that the user manipulates. However, we will consider structural elements, such as the groups that aggregate objects or the viewing transforms that map virtual worlds to screen coordinates as objects as well.

For a graphical object, there are two "sides" which we must consider. On one hand is what the programmer "sees," the object's internal representation. An important part of this are the parameters that determine the configuration of the object. Each object stores this set of numbers as its *state vector.*

To the user, the graphical object should appear as a graphical object. We assume that the user is interested in the graphical entity, not in the internal data structures used by the programmer. For any object, there are many attributes that may be of interest to the user, or to other objects in the program for that matter.

Ideally, we would like to think of a graphical object as a sealed box. Inside is the programmer's internal representation, including the state vector. To the outside world, all that is visible are the many attributes which other parts of the program, or the user, may want to observe. Our desire to think this way leads us to draw graphical objects schematically as Figure 1.2. The central notion is that the state is internal to the object and the object's "outputs" are its attributes. How the object computes these attributes is the concern of the object itself, not the outside world.

The state vector of an object fully specifies its configuration. Therefore, any attribute of the object must be a function of these variables. This function must be known, otherwise it would be impossible to compute the value of the attribute.

A graphical object may know how to compute many attributes. The set of attributes

of an object is not necessarily fixed — an object may have many attributes, and new attributes may be created in response to the needs of some other part of the system or the user. The schematic of Figure 1.2 may be slightly misleading in that it should not imply that the depicted outputs are a fixed, small set.

We will call the outputs of graphical objects *connectors*. As the name implies, these are the sockets into which the outside world will connect to the object. A connector is an attribute that an object provides for the outside world to access. Throughout this thesis, the notion of connector will be both a conceptual idea as well as a data structure that realizes it.

1.3.2 Compound Objects and Dependencies

Many attributes can be computed as functions of other attributes, rather than from inside the object. For example, if we wish to know the length of a line segment, this attribute could be computed as a function of the positions of endpoints. Therefore, if the line segment did not know how to produce its length as a connector, we might create a special ruler object that looks at the positions of two points and "connect" it to the endpoint outputs of the line segment.

An important notion in the ruler example is that the ruler object takes as its "inputs" the "outputs" of another object. The ruler measures the distance between two points, without concern for what these points are. This is significant for three reasons:

- It means that the objects, such as the line segment, can be extended to have new behaviors without being internally modified.
- It means that we need only one type of ruler, no matter how many different types of objects we might be measuring.
- We are not necessarily restricted to points on a single object. Instead we could measure the distance between two points on two different objects.

Objects like the ruler have inputs that plug in to the output connectors of other graphical objects. Considering such dependencies leads us to draw schematic diagrams such as Figure 1.3. The outputs of the connective objects are attributes just like the outputs of the simpler objects. The distance output of the ruler should be a first-class citizen, just as the position outputs of the line segments. Like the outputs on simpler object, the connectors on the ruler object's outputs are also functions of the state vector, except that they are potentially functions of the state vector of the entire model (which we will call the *global* state vector), rather than just the state vector of a single object. The function that determines the attribute's value can be built by composition: first computing the values of the inputs and then using these values as the inputs to a function which computes the distance.

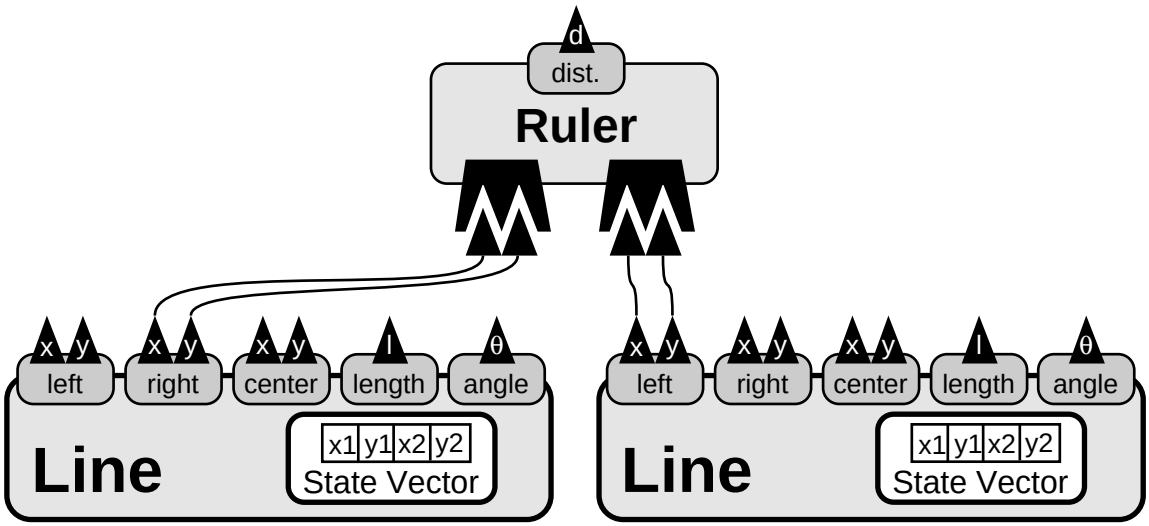

Figure 1.3: Compound objects are composed by plugging objects' connectors into sockets, like wiring together a circuit. A standardized protocol allows independence in wiring.

This picture emphasizes an important notion in the thesis: the idea of plugging objects into the "outputs" of other objects. The facility to dynamically plug and unplug such connections in response to user actions or other system events is an essential part of the differential approach, and will figure prominently in the design of the machinery to realize it.

The key element for creating the vision of snap-together objects in the differential approach is a standard protocol for the outputs so that anything can be plugged in. Since the connector outputs are primarily functions, the aggregate connection operation is function composition: building more complicated functions from simpler pieces. By supporting this operation in a dynamic environment, the machinery to realize the differential approach can permit the needed plugging and unplugging.

Compound objects, like the ruler, can come in many forms. Typically, they are used to compute aggregate properties of many different objects. For example, the distance between two points, or the relative orientation of two line segments. They may also be used to compute conversions, for example from degrees to radians. More complex attributes can also be built this way, for example, we might compute the position of a shadow as a compound operation that takes the position of a point, the position of a light source, and the position of the floor as its inputs. Flexibility in building new types of attribute outputs is a useful feature of the differential approach.

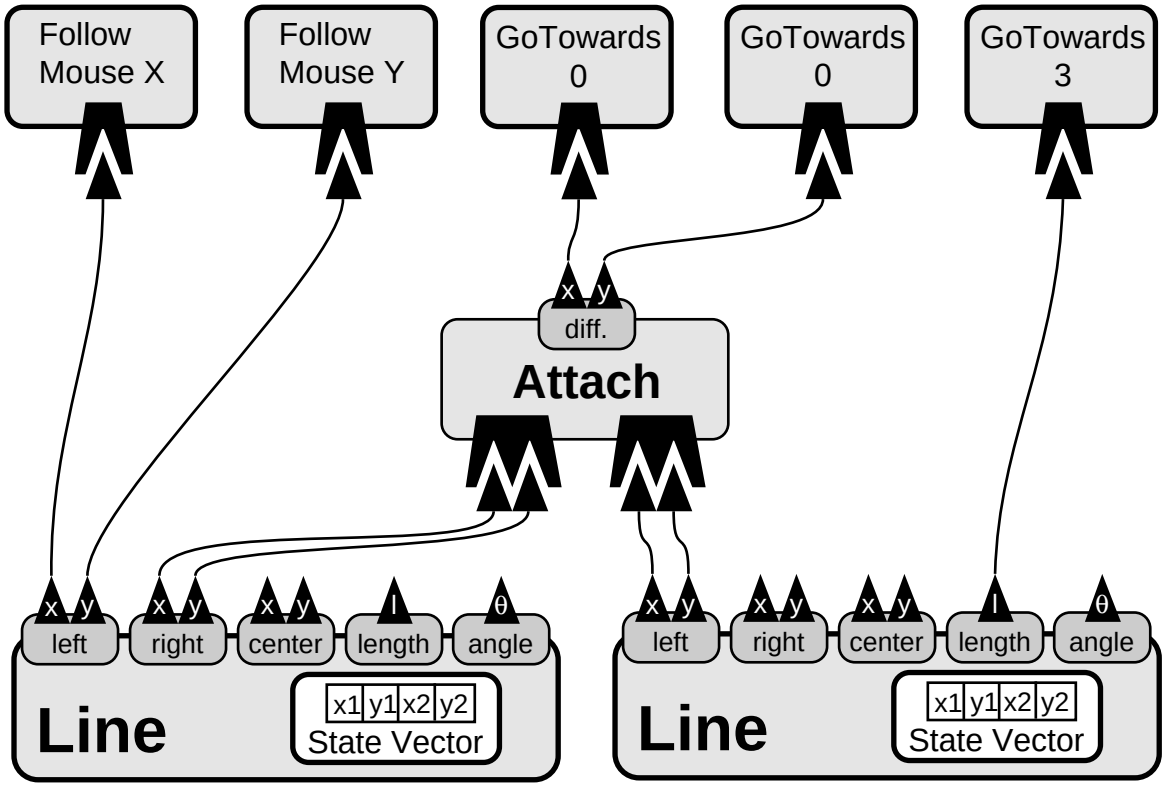

Figure 1.4: Objects are manipulated by attaching controllers to their connectors. A controller specifies how the value of a connector should be changing. Controllers can be plugged into any connector. This diagram represents a model with two line segments that are attached. One segment has its length constrained, while the other is being dragged.

1.3.3 Control of Graphical Objects

Since the attributes are the only view of an object that the programmer is given, it follows that these attributes must also serve as the handles by which the object is controlled. The vision of the differential approach is that any attribute output should be able to serve as a mechanism to control the object, and that these controls should be able to be freely applied as needed in any desired combination. Thus, any output should also be able to serve as an input.

Our notion of using an output as an input can be best discussed by introducing another kind of special object, the *controller.* A controller is a simple object that plugs into a connector and specifies what behavior the outside world desires from it. With this final abstraction, we are led to draw schematics such as Figure 1.4.

With the abstractions in place, we can now examine Figure 1.4 to see the mathematical constraint problem. We have specified the outputs of the functions that compute

the attributes being used as controls, and must determine the inputs to these functions (the value of the state vector) to achieve the desired values.

As discussed in Section 1.2, we cannot solve this constraint problem directly. Instead, we will solve it differentially. This means that rather than specifying desired values for attributes, controllers specify how they should be changing over time. A controller specifies a rate of change for the attribute it is connected to.

It is important to notice that the controllers cannot instantaneously affect the values of the connectors they control, nor the state variables of the objects. Instead, they specify how those connectors are changing, and over time, those changes will take effect. This implies that there is a continuous flow of time over which the controllers can act. At discrete instants, the set of active controllers may be altered, but values cannot be changed.

What a controller can do is quite limited: it can simply specify the desired rate of change of an attribute. The diversity of interaction techniques comes not from diversity in the types of controllers, but rather, from the way they are applied. Interesting interaction techniques result from:

- attaching controllers to interesting attributes;
- connecting controllers at interesting times;
- using controllers in interesting combinations.

Interaction techniques are defined by controlling connectors over time. For example, to drag a point, the connector that computes the point's position is connected to a controller when the mouse button is pressed to initiate the drag, and the controller is removed when the mouse button is released. Similarly, a mechanical connection between two points is created by using an object which computes the displacement between two points and creating a controller which drives the displacement to zero.

The differential approach provides a basic set of abstractions from which interfaces and interaction techniques can be built. The ability to wire together attributes and attach controllers to them provides machinery that can be applied in a wide variety of manners.

One interface style which is enabled by the differential approach is to provide the abstractions directly to the user, permitting them to mix and match controls as needed. For example, in the lamp demonstration, the user would be permitted to grab and drag many points involving the lamp, including the light's target, the bulb, and the corners of the base. Attributes which are not positional, such as joint angles or bulb brightness, might be connected to sliders. The user could configure the lamp by manipulating any of these controls, or by constraining their values. Controls are mixed-and-matched by manipulating or locking their values. This interface style is similar to a traditional constraint-based interface. Many of the issues which make constraint-based interfaces difficult to design must be addressed, such as how to present the palette of options to the user effectively.

Another way that the differential abstractions may be employed is to build interaction techniques which are more similar to the traditional direct manipulation interfaces. An example is the 3D translation widget discussed in Section 8.3.6. To the user, the translation handles appear as they do in other systems which provide them. However, this interaction technique can be concisely described by defining sets of controllers during dragging. While the differential approach is merely used to recreate an existing technique in such cases, it does have some interesting benefits. The differential approach addresses the difficult question of how to define such interesting behaviors in a way that is parameter independent, and easy to generalize to other controllers.

1.3.4 Impact on Application Architecture

Just as the differential approach frees the user from worrying about the object representations, it can also hide such parameterizations from the programmers of graphical applications, helping to foster encapsulation. Objects merely expose mathematical functional outputs for attributes that other pieces of the system may be interested in. The program manipulates the object by placing constraints and controls on these ports, and the differential solving mechanism takes care of adjusting the parameters accordingly.

The solving mechanism needs very little information about the functions that are being constrained and controlled. This means that objects need not expose much information about the functions they provide. It also simplifies the composition of functions from pieces, such as object outputs. This allows creation of a utility which permits functions to be defined dynamically, for example in response to user actions. The core functionality of the differential approach, the ability to define functions and place constraints and controls on them, can be built in a general purpose manner.

The general protocol for connecting the outputs of objects permits the creation of general purpose objects, constraints, and interaction techniques. Objects can provide mathematical ports without regard for what will "plug-in" to these ports. Constraints and interaction techniques can be defined in terms of types of outputs, without regard for the objects that are being connected to. For example, we define graphical objects that produce outputs that are the positions of points, and define constraints and interaction techniques in terms of point position outputs.

1.4 Thesis Roadmap

This thesis introduces the differential approach, presents techniques to realize it, and provides examples to illustrate its power and viability. Following this introduction, the thesis proceeds to review some relevant related work.

Chapter 3 introduces the basic set of mathematical techniques required to realize

the differential approach. The methods treat manipulation as equation solving. This problem is handled differentially to make it feasible to solve. The fundamental computation is solving a constrained optimization problem to compute how the parameters of objects are changing given the rates of change of the controls. Basic methods for solving these constrained optimization problems are developed and extended to handle under- and over- constrained cases. The chapter also considers how to use the computed rates of change to actually create the motion, a problem of solving ordinary differential equations from initial values. The chapter concludes with a simple example, worked through in detail.

In order to use numerical techniques in an interactive system, there are two central challenges that must be faced. The computations must be made to go fast enough, and the computations must be defined dynamically in response to the users actions. These issues are considered in Chapter 4 and Chapter 5 respectively. Chapter 4 considers methods to achieve the needed performance in such solving. After analyzing the computational bottlenecks of the approach, a variety of methods are presented to enhance performance. One key element is exploiting the inherent sparsity of systems of equations to be solved. Other techniques include solving smaller problems while still giving the user the illusion that the system is solving a larger problem, and trading unneeded accuracy for speed.

Chapter 5 considers the task of dynamically defining functions in a way that they can be rapidly evaluated with their derivatives. A tool called Snap-Together Mathematics that allows functions to be built dynamically from smaller pieces is presented. Snap-Together Mathematics is an important element of the differential approach because it provides the software structure for dynamically mixing and matching controls, and provides a mechanism for encapsulating the mathematics of the approach.

With the basic machinery in place, Chapter 6 considers how the tools are applied to create interaction techniques. It defines the set of abstractions provided to interface designers by the approach, and describes how the differential notion of time is different than what is commonly used in interactive-systems programming. The chapter provides some basic examples of how the abstractions are employed, and provides some extensions to the basic differential techniques to permit such things as inequality constraints.

Chapter 7 discusses how the differential approach can be encapsulated into a graphics toolkit. The Bramble toolkit was designed to aid in the development of graphical editing applications with the differential approach. Various elements of the toolkit are described, with an emphasis on how it supports the differential approach.

Chapter 8 describes interaction techniques built using the abstractions of the differential approach. It begins by discussing some basic strategies. It then provides concrete examples of techniques to address various interaction tasks. In addition to several novel interaction techniques, many previous techniques are recreated, in order to show how the Differential Approach can be applied to these problems.

Chapter 9 presents some example applications built with the approach. The applications serve to demonstrate the viability of the approach and to give some idea of its promise in constructing tools for users. Chapter 10 concludes the thesis by summarizing the contributions, evaluating the various contents, and suggesting some directions for future work.

1.5 The Thesis

It is the premise of this thesis that:

- The numerical and graphical performance of modern processors can be applied to address issues in graphical manipulation.
- A *differential approach* to graphical interaction provides a systematic implementation of direct manipulation. This approach allows a system to provide users with a broad class of interactive controls that can be freely combined, yet preserves direct manipulation, so it does not suffer from the drawbacks of other previous approaches.
- Mathematical techniques to realize the differential approach can be provided, and that these techniques can be realized such that the issues of interactive systems are addressed. In particular, methods permit the computations to be defined dynamically in response to user actions and to be performed sufficiently fast on current generation hardware.
- The techniques of the differential approach can be encapsulated, providing a set of abstractions with which to build interfaces as well as a general purpose implementation.
- The differential approach can have a positive impact on the way that interaction techniques are developed and that interactive systems are constructed, by helping separate manipulation from representation and by enabling general purpose constraints and interaction techniques.
- The differential approach can lead to interesting new interaction techniques and applications, but can also serve as a substrate for implementing existing popular interaction techniques.

1.5.1 Contributions

The contributions of this thesis are detailed in the final chapter. Briefly and generally, the contributions of this thesis are (in the order they will be presented in the thesis):

- To introduce a systematic approach to graphical interaction based on the use of numerical non-linear constraint techniques, which I call the differential approach.
- To present mathematical techniques for solving the particular constrained optimization problems encountered in using the differential approach.
- To provide techniques to implement these mathematical techniques that address the pragmatic needs of interactive systems.
- To provide a toolkit that encapsulates the differential approach, providing its features to application developers while shielding them from the details of its implementation.
- To provide new interaction techniques and examples to address problems faced by users of interactive graphical applications, and to show how these techniques fit in the context of graphical applications.
- To provide example applications demonstrating the viability of the approach.